1 Big O

Time Complexity

The time complexity of an algorithm estimates how much time the algorithm will use for a given input. The time complexity is denoted by $\boldsymbol{O(\cdots)}$ where the three dots represent some function based on the input size, usually denoted by n.

Common Time Complexities

$\boldsymbol{O(1)}$ *Constantant time.* The running time does not depend on the input size. A typical constant-time is a direct formula that calculates the answer.

$\boldsymbol{O(\log n)}$ The *logarithmic* often halves the input size at each step. The running time of such an algorithm is logarithmic, because $\log_2 n$ equals the number of times n must be divided by 2 to get 1.

$\boldsymbol{O(\sqrt{n})}$ A *square root* algorithm is slower than $O(\log n)$ but faster than $O(n)$. A special property of square roots is that $\sqrt{n} = n/\sqrt{n}$, so n elements can be divided into $O(\sqrt{n})$ blocks of $O(\sqrt{n})$ elements.

$\boldsymbol{O(n)}$ A linear algorithm goes through the input a constant number of times. This is often the best possible time complexity, because it is usually necessary to access each input element at least once before reporting the answer.

$\boldsymbol{O(n \log n)}$ This time complexity often indicates that the algorithm sorts the input, because the time complexity of efficient sorting algorithms is $O(n \log n)$. Another possibility is that the algorithm uses a data structure where each operation takes $O(\log n)$ time.

$\boldsymbol{O(n^2)}$ A quadratic algorithm often contains two nested loops. It is possible to go through all pairs of the input elements in $O(n^2)$ time.

$\boldsymbol{O(n^3)}$ A cubic algorithm often contains three nested loops. It is possible to go through all triplets of the input elements in $O(n^3)$ time.

$\boldsymbol{O(2^n)}$ This time complexity often indicates that the algorithm iterates through all subsets of the input elements. For example, the subsets of $\{1, 2, 3\}$ are $\emptyset$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$, and $\{1, 2, 3\}$.

$\boldsymbol{O(n!)}$ This time complexity often indicates that the algorithm iterates through all permutations of the input elements. For example, the permutations of $\{1, 2, 3\}$ are (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), and (3,2,1).